

autogluon_each_location

October 16, 2023

```
[6]: # config

label = 'y'
metric = 'mean_absolute_error'
time_limit = 60*60*2
presets = 'best_quality'

do_drop_ds = True
# hour, dayofweek, dayofmonth, month, year
use_dt_attrs = ["hour"]
use_estimated_diff_attr = False
use_is_estimated_attr = True

use_groups = False
n_groups = 8

auto_stack = True
num_stack_levels = 2
num_bag_folds = 8

use_tune_data = False
use_test_data = False
tune_and_test_length = 24*30*3 # 3 months from end
holdout_frac = None
use_bag_holdout = False # Enable this if there is a large gap between score_val_
    ↪ and score_test in stack models.

sample_weight = 'sample_weight' #None
weight_evaluation = True
sample_weight_estimated = 2

run_analysis = True

[7]: import pandas as pd
import numpy as np
```

```

import warnings
warnings.filterwarnings("ignore")

def feature_engineering(X):
    # shift all columns with "1h" in them by 1 hour, so that for index 16:00,
    # we have the values from 17:00
    # but only for the columns with "1h" in the name
    #X_shifted = X.filter(regex="\dh").shift(-1, axis=1)
    #print(f"Number of columns with 1h in name: {X_shifted.columns}")

    columns = ['clear_sky_energy_1h:J', 'diffuse_rad_1h:J', 'direct_rad_1h:J',
               'fresh_snow_12h:cm', 'fresh_snow_1h:cm', 'fresh_snow_24h:cm',
               'fresh_snow_3h:cm', 'fresh_snow_6h:cm']

    X_shifted = X[X.index.minute==0][columns].copy()
    # loop through all rows and check if index + 1 hour is in the index, if so
    # get that value, else nan
    count1 = 0
    count2 = 0
    for i in range(len(X_shifted)):
        if X_shifted.index[i] + pd.Timedelta('1 hour') in X.index:
            count1 += 1
            X_shifted.iloc[i] = X.loc[X_shifted.index[i] + pd.Timedelta('1
            hour')][columns]
        else:
            count2 += 1
            X_shifted.iloc[i] = np.nan

    print("COUNT1", count1)
    print("COUNT2", count2)

    X_old_unshifted = X[X.index.minute==0][columns]
    # rename X_old_unshifted columns to have _not_shifted at the end
    X_old_unshifted.columns = [f"{col}_not_shifted" for col in X_old_unshifted.
    columns]

    # put the shifted columns back into the original dataframe
    #X[columns] = X_shifted[columns]

    date_calc = None
    if "date_calc" in X.columns:

```

```

        date_calc = X[X.index.minute == 0]['date_calc']

        # resample to hourly
        X = X.resample('H').mean()

        X[columns] = X_shifted[columns]
        #X[X_old_unshifted.columns] = X_old_unshifted

        if date_calc is not None:
            X['date_calc'] = date_calc

        return X

def fix_X(X, name):
    # Convert 'date_forecast' to datetime format and replace original column
    # with 'ds'
    X['ds'] = pd.to_datetime(X['date_forecast'])
    X.drop(columns=['date_forecast'], inplace=True, errors='ignore')
    X.sort_values(by='ds', inplace=True)
    X.set_index('ds', inplace=True)

    X = feature_engineering(X)

    return X

def handle_features(X_train_observed, X_train_estimated, X_test, y_train):
    X_train_observed = fix_X(X_train_observed, "X_train_observed")
    X_train_estimated = fix_X(X_train_estimated, "X_train_estimated")
    X_test = fix_X(X_test, "X_test")

    # add sample weights, which are 1 for observed and 3 for estimated
    X_train_observed["sample_weight"] = 1
    X_train_estimated["sample_weight"] = sample_weight_estimated
    X_test["sample_weight"] = sample_weight_estimated

    y_train['ds'] = pd.to_datetime(y_train['time'])
    y_train.drop(columns=['time'], inplace=True)
    y_train.sort_values(by='ds', inplace=True)
    y_train.set_index('ds', inplace=True)

```

```

    return X_train_observed, X_train_estimated, X_test, y_train

def preprocess_data(X_train_observed, X_train_estimated, X_test, y_train,
    ↪location):
    # convert to datetime
    X_train_observed, X_train_estimated, X_test, y_train =
    ↪handle_features(X_train_observed, X_train_estimated, X_test, y_train)

    if use_estimated_diff_attr:
        X_train_observed["estimated_diff_hours"] = 0
        X_train_estimated["estimated_diff_hours"] = (X_train_estimated.index -
    ↪pd.to_datetime(X_train_estimated["date_calc"])).dt.total_seconds() / 3600
        X_test["estimated_diff_hours"] = (X_test.index - pd.
    ↪to_datetime(X_test["date_calc"])).dt.total_seconds() / 3600

        X_train_estimated["estimated_diff_hours"] =
    ↪X_train_estimated["estimated_diff_hours"].astype('int64')
        # the filled once will get dropped later anyways, when we drop y nans
        X_test["estimated_diff_hours"] = X_test["estimated_diff_hours"].
    ↪fillna(-50).astype('int64')

    if use_is_estimated_attr:
        X_train_observed["is_estimated"] = 0
        X_train_estimated["is_estimated"] = 1
        X_test["is_estimated"] = 1

    # drop date_calc
    X_train_estimated.drop(columns=['date_calc'], inplace=True)
    X_test.drop(columns=['date_calc'], inplace=True)

    y_train["y"] = y_train["pv_measurement"].astype('float64')
    y_train.drop(columns=['pv_measurement'], inplace=True)
    X_train = pd.concat([X_train_observed, X_train_estimated])

    # clip all y values to 0 if negative
    y_train["y"] = y_train["y"].clip(lower=0)

    X_train = pd.merge(X_train, y_train, how="inner", left_index=True,
    ↪right_index=True)

```

```

    # print number of nans in sample_weight
    print(f"Number of nans in sample_weight: {X_train['sample_weight'].isna().
↪sum()}")
    # print number of nans in y
    print(f"Number of nans in y: {X_train['y'].isna().sum()}")

    X_train["location"] = location
    X_test["location"] = location

    return X_train, X_test
# Define locations
locations = ['A', 'B', 'C']

X_trains = []
X_tests = []
# Loop through locations
for loc in locations:
    print(f"Processing location {loc}...")
    # Read target training data
    y_train = pd.read_parquet(f'{loc}/train_targets.parquet')

    # Read estimated training data and add location feature
    X_train_estimated = pd.read_parquet(f'{loc}/X_train_estimated.parquet')

    # Read observed training data and add location feature
    X_train_observed = pd.read_parquet(f'{loc}/X_train_observed.parquet')

    # Read estimated test data and add location feature
    X_test_estimated = pd.read_parquet(f'{loc}/X_test_estimated.parquet')

    # Preprocess data
    X_train, X_test = preprocess_data(X_train_observed, X_train_estimated,
↪X_test_estimated, y_train, loc)

    X_trains.append(X_train)
    X_tests.append(X_test)

# Concatenate all data and save to csv
X_train = pd.concat(X_trains)
X_test = pd.concat(X_tests)

```

Processing location A...

COUNT1 29667

COUNT2 1

COUNT1 4392

COUNT2 2

```

COUNT1 702
COUNT2 18
Number of nans in sample_weight: 0
Number of nans in y: 0
Processing location B...
COUNT1 29232
COUNT2 1
COUNT1 4392
COUNT2 2
COUNT1 702
COUNT2 18
Number of nans in sample_weight: 0
Number of nans in y: 4
Processing location C...
COUNT1 29206
COUNT2 1
COUNT1 4392
COUNT2 2
COUNT1 702
COUNT2 18
Number of nans in sample_weight: 0
Number of nans in y: 6059

```

1 Feature engineering

```

[8]: import numpy as np
import pandas as pd

X_train.dropna(subset=['y'], inplace=True)

for attr in use_dt_attrs:
    X_train[attr] = getattr(X_train.index, attr)
    X_test[attr] = getattr(X_test.index, attr)

print(X_train.head())

if use_groups:
    # fix groups for cross validation
    locations = X_train['location'].unique() # Assuming 'location' is the name
    ↪ of the column representing locations

    grouped_dfs = [] # To store data frames split by location

```

```

# Loop through each unique location
for loc in locations:
    loc_df = X_train[X_train['location'] == loc]

    # Sort the DataFrame for this location by the time column
    loc_df = loc_df.sort_index()

    # Calculate the size of each group for this location
    group_size = len(loc_df) // n_groups

    # Create a new 'group' column for this location
    loc_df['group'] = np.repeat(range(n_groups),
    ↪repeats=[group_size]*(n_groups-1) + [len(loc_df) - group_size*(n_groups-1)])

    # Append to list of grouped DataFrames
    grouped_dfs.append(loc_df)

# Concatenate all the grouped DataFrames back together
X_train = pd.concat(grouped_dfs)
X_train.sort_index(inplace=True)
print(X_train["group"].head())

to_drop = ["snow_drift:idx", "snow_density:kgm3", "wind_speed_w_1000hPa:ms",
    ↪"dew_or_rime:idx", "prob_rime:p", "fresh_snow_12h:cm", "fresh_snow_24h:cm"]

X_train.drop(columns=to_drop, inplace=True)
X_test.drop(columns=to_drop, inplace=True)

X_train.to_csv('X_train_raw.csv', index=True)
X_test.to_csv('X_test_raw.csv', index=True)

```

	absolute_humidity_2m:gm3	air_density_2m:kgm3	\
ds			
2019-06-02 22:00:00	7.700	1.22825	
2019-06-02 23:00:00	7.700	1.22350	
2019-06-03 00:00:00	7.875	1.21975	
2019-06-03 01:00:00	8.425	1.21800	
2019-06-03 02:00:00	8.950	1.21800	

	ceiling_height_agl:m	clear_sky_energy_1h:J	\
ds			
2019-06-02 22:00:00	1728.949951	0.000000	
2019-06-02 23:00:00	1689.824951	0.000000	

2019-06-03 00:00:00	1563.224976	0.000000
2019-06-03 01:00:00	1283.425049	6546.899902
2019-06-03 02:00:00	1003.500000	102225.898438

	clear_sky_rad:W	cloud_base_agl:m	dew_or_rime:idx	\
ds				
2019-06-02 22:00:00	0.00	1728.949951	0.0	
2019-06-02 23:00:00	0.00	1689.824951	0.0	
2019-06-03 00:00:00	0.00	1563.224976	0.0	
2019-06-03 01:00:00	0.75	1283.425049	0.0	
2019-06-03 02:00:00	23.10	1003.500000	0.0	

	dew_point_2m:K	diffuse_rad:W	diffuse_rad_1h:J	...	\
ds					
2019-06-02 22:00:00	280.299988	0.000	0.000000	...	
2019-06-02 23:00:00	280.299988	0.000	0.000000	...	
2019-06-03 00:00:00	280.649994	0.000	0.000000	...	
2019-06-03 01:00:00	281.674988	0.300	7743.299805	...	
2019-06-03 02:00:00	282.500000	11.975	60137.601562	...	

	visibility:m	wind_speed_10m:ms	wind_speed_u_10m:ms	\
ds				
2019-06-02 22:00:00	40386.476562	3.600	-3.575	
2019-06-02 23:00:00	33770.648438	3.350	-3.350	
2019-06-03 00:00:00	13595.500000	3.050	-2.950	
2019-06-03 01:00:00	2321.850098	2.725	-2.600	
2019-06-03 02:00:00	11634.799805	2.550	-2.350	

	wind_speed_v_10m:ms	wind_speed_w_1000hPa:ms	\
ds			
2019-06-02 22:00:00	-0.500	0.0	
2019-06-02 23:00:00	0.275	0.0	
2019-06-03 00:00:00	0.750	0.0	
2019-06-03 01:00:00	0.875	0.0	
2019-06-03 02:00:00	0.925	0.0	

	sample_weight	is_estimated	y	location	hour
ds					
2019-06-02 22:00:00	1	0	0.00	A	22
2019-06-02 23:00:00	1	0	0.00	A	23
2019-06-03 00:00:00	1	0	0.00	A	0
2019-06-03 01:00:00	1	0	0.00	A	1
2019-06-03 02:00:00	1	0	19.36	A	2

[5 rows x 50 columns]


```

[9]: from autogluon.tabular import TabularDataset, TabularPredictor
from autogluon.timeseries import TimeSeriesDataFrame
import numpy as np
train_data = TabularDataset('X_train_raw.csv')
# set group column of train_data be increasing from 0 to 7 based on time, the
  ↳ first 1/8 of the data is group 0, the second 1/8 of the data is group 1, etc.
train_data['ds'] = pd.to_datetime(train_data['ds'])
train_data = train_data.sort_values(by='ds')

# # print size of the group for each location
# for loc in locations:
#     print(f"Location {loc}:")
#     print(train_data[train_data["location"] == loc].groupby('group').size())

# get end date of train data and subtract 3 months
split_time = pd.to_datetime(train_data["ds"]).max() - pd.
  ↳ Timedelta(hours=tune_and_test_length)
train_set = TabularDataset(train_data[train_data["ds"] < split_time])
test_set = TabularDataset(train_data[train_data["ds"] >= split_time])
if use_groups:
    test_set = test_set.drop(columns=['group'])

if do_drop_ds:
    train_set = train_set.drop(columns=['ds'])
    test_set = test_set.drop(columns=['ds'])
    train_data = train_data.drop(columns=['ds'])

def normalize_sample_weights_per_location(df):
    for loc in locations:
        loc_df = df[df["location"] == loc]
        loc_df["sample_weight"] = loc_df["sample_weight"] /
  ↳ loc_df["sample_weight"].sum() * loc_df.shape[0]
        df[df["location"] == loc] = loc_df
    return df

tuning_data = None
if use_tune_data:
    train_data = train_set
    if use_test_data:
        # split test_set in half, use first half for tuning
        tuning_data, test_data = [], []
        for loc in locations:
            loc_test_set = test_set[test_set["location"] == loc]
            loc_tuning_data = loc_test_set.iloc[:len(loc_test_set)//2]
            loc_test_data = loc_test_set.iloc[len(loc_test_set)//2:]

```

```

        tuning_data.append(loc_tuning_data)
        test_data.append(loc_test_data)
        tuning_data = pd.concat(tuning_data)
        test_data = pd.concat(test_data)
        print("Shapes of tuning and test", tuning_data.shape[0], test_data.
↪shape[0], tuning_data.shape[0] + test_data.shape[0])

    else:
        tuning_data = test_set
        print("Shape of tuning", tuning_data.shape[0])

        # ensure sample weights for your tuning data sum to the number of rows in
↪the tuning data.
        tuning_data = normalize_sample_weights_per_location(tuning_data)

else:
    if use_test_data:
        train_data = train_set
        test_data = test_set
        print("Shape of test", test_data.shape[0])

    # ensure sample weights for your training (or tuning) data sum to the number of
↪rows in the training (or tuning) data.
    train_data = normalize_sample_weights_per_location(train_data)
    if use_test_data:
        test_data = normalize_sample_weights_per_location(test_data)

```

```

[10]: if run_analysis:
        import autogluon.eda.auto as auto
        auto.dataset_overview(train_data=train_data, test_data=test_set, label="y",
↪sample=None)

```

train_data dataset summary

	count	unique top	freq	mean \
absolute_humidity_2m:gm3	92951	760		6.017393
air_density_2m:kgm3	92951	1374		1.255435
ceiling_height_agl:m	76534	63118		2888.30088
clear_sky_energy_1h:J	92945	48602	515183.641476	
clear_sky_rad:W	92951	20312	143.098015	
cloud_base_agl:m	86213	63893	1735.995178	
dew_point_2m:K	92951	2006	275.237958	
diffuse_rad:W	92951	11222	39.395201	
diffuse_rad_1h:J	92945	48552	142196.335278	
direct_rad:W	92951	14417	50.24518	
direct_rad_1h:J	92945	41885	180744.819009	
effective_cloud_cover:p	92951	5713	67.08644	

elevation:m	92951	3		11.401739
fresh_snow_1h:cm	92945	39		0.009641
fresh_snow_3h:cm	92945	70		0.029042
fresh_snow_6h:cm	92945	96		0.058143
hour	93023	24		11.501338
is_day:idx	92951	5		0.483303
is_estimated	93023	2		0.118218
is_in_shadow:idx	92951	5		0.564284
location	93023	3	A 34085	
msl_pressure:hPa	92951	3733		1009.502496
precip_5min:mm	92951	271		0.005657
precip_type_5min:idx	92951	15		0.084348
pressure_100m:hPa	92951	3760		995.818828
pressure_50m:hPa	92951	3809		1001.949597
rain_water:kgm2	92951	39		0.009566
relative_humidity_1000hPa:p	92951	3837		73.670589
sample_weight	93023	6		1.0
sfc_pressure:hPa	92951	3817		1008.107684
snow_depth:cm	92951	491		0.193164
snow_melt_10min:mm	92951	66		0.000273
snow_water:kgm2	92951	162		0.090299
sun_azimuth:d	92951	88092		179.648569
sun_elevation:d	92951	76035		-1.206875
super_cooled_liquid_water:kgm2	92951	53		0.056897
t_1000hPa:K	92951	1994		279.430664
total_cloud_cover:p	92951	5608		73.692549
visibility:m	92951	91240		33025.01299
wind_speed_10m:ms	92951	596		3.038167
wind_speed_u_10m:ms	92951	999		0.664565
wind_speed_v_10m:ms	92951	850		0.685095
y	93023	12379		287.022737

	std	min	25% \
absolute_humidity_2m:gm3	2.711862	0.5	4.025
air_density_2m:kgm3	0.036567	1.13925	1.23025
ceiling_height_agl:m	2536.68272	27.8	1087.60625
clear_sky_energy_1h:J	820542.211615	0.0	0.0
clear_sky_rad:W	227.959967	0.0	0.0
cloud_base_agl:m	1809.297261	27.5	591.925
dew_point_2m:K	6.829573	247.425	270.75
diffuse_rad:W	60.518576	0.0	0.0
diffuse_rad_1h:J	215920.02629	0.0	0.0
direct_rad:W	112.91716	0.0	0.0
direct_rad_1h:J	401738.480227	0.0	0.0
effective_cloud_cover:p	34.269564	0.0	42.0
elevation:m	7.877236	6.0	6.0
fresh_snow_1h:cm	0.112651	0.0	0.0
fresh_snow_3h:cm	0.280779	0.0	0.0

fresh_snow_6h:cm	0.481544	0.0	0.0
hour	6.920154	0.0	6.0
is_day:idx	0.485974	0.0	0.0
is_estimated	0.322868	0.0	0.0
is_in_shadow:idx	0.483166	0.0	0.0
location			
mssl_pressure:hPa	13.085625	944.375	1001.4
precip_5min:mm	0.029169	0.0	0.0
precip_type_5min:idx	0.330071	0.0	0.0
pressure_100m:hPa	13.004987	929.975	987.775
pressure_50m:hPa	13.063835	935.75	993.85
rain_water:kgm2	0.041121	0.0	0.0
relative_humidity_1000hPa:p	14.229107	19.575	64.2
sample_weight	0.288512	0.885256	0.885256
sfc_pressure:hPa	13.124815	941.55	999.975
snow_depth:cm	1.253925	0.0	0.0
snow_melt_10min:mm	0.004249	0.0	0.0
snow_water:kgm2	0.237841	0.0	0.0
sun_azimuth:d	97.282534	6.983	94.67875
sun_elevation:d	23.970707	-49.932	-18.59975
super_cooled_liquid_water:kgm2	0.105794	0.0	0.0
t_1000hPa:K	6.515625	258.025	274.9
total_cloud_cover:p	34.021943	0.0	53.225
visibility:m	17913.98226	132.375	16862.7995
wind_speed_10m:ms	1.760291	0.025	1.675
wind_speed_u_10m:ms	2.802007	-7.225	-1.35
wind_speed_v_10m:ms	1.878808	-8.4	-0.575
y	766.411327	-0.0	0.0

	50%	75%	max	dtypes \
absolute_humidity_2m:gm3	5.45	7.825	17.35	float64
air_density_2m:kgm3	1.255	1.2785	1.441	float64
ceiling_height_agl:m	1887.8875	3988.4125	12294.901	float64
clear_sky_energy_1h:J	4551.0	778482.3	3006697.2	float64
clear_sky_rad:W	1.65	216.8	835.65	float64
cloud_base_agl:m	1164.525	2079.25	11673.725	float64
dew_point_2m:K	274.975	280.5	293.625	float64
diffuse_rad:W	0.925	65.275	334.75	float64
diffuse_rad_1h:J	9952.6	236534.8	1182265.4	float64
direct_rad:W	0.0	29.3	683.4	float64
direct_rad_1h:J	0.0	113395.3	2445897.0	float64
effective_cloud_cover:p	79.95	98.637497	100.0	float64
elevation:m	7.0	24.0	24.0	float64
fresh_snow_1h:cm	0.0	0.0	7.1	float64
fresh_snow_3h:cm	0.0	0.0	20.6	float64
fresh_snow_6h:cm	0.0	0.0	34.0	float64
hour	12.0	17.0	23.0	int64
is_day:idx	0.25	1.0	1.0	float64

is_estimated	0.0	0.0	1.0	int64
is_in_shadow:idx	1.0	1.0	1.0	float64
location				object
msl_pressure:hPa	1010.35	1018.55	1044.1	float64
precip_5min:mm	0.0	0.0	0.6225	float64
precip_type_5min:idx	0.0	0.0	5.0	float64
pressure_100m:hPa	996.75	1004.925	1030.875	float64
pressure_50m:hPa	1002.85	1011.05	1037.25	float64
rain_water:kgm2	0.0	0.0	1.1	float64
relative_humidity_1000hPa:p	76.0	85.05	100.0	float64
sample_weight	0.89831	0.900598	1.801196	float64
sfc_pressure:hPa	1009.0	1017.2	1043.725	float64
snow_depth:cm	0.0	0.0	18.2	float64
snow_melt_10min:mm	0.0	0.0	0.18	float64
snow_water:kgm2	0.0	0.1	5.65	float64
sun_azimuth:d	179.97975	264.41998	348.48752	float64
sun_elevation:d	-0.8645	15.25075	49.94375	float64
super_cooled_liquid_water:kgm2	0.0	0.1	1.375	float64
t_1000hPa:K	278.65002	283.95	303.25	float64
total_cloud_cover:p	93.05	99.9	100.0	float64
visibility:m	36846.176	48308.9875	75489.33	float64
wind_speed_10m:ms	2.7	4.05	13.275	float64
wind_speed_u_10m:ms	0.3	2.5	11.2	float64
wind_speed_v_10m:ms	0.725	1.875	8.825	float64
y	0.0	172.92	5733.42	float64

	missing_count	missing_ratio	raw_type	\
absolute_humidity_2m:gm3	72	0.000774	float	
air_density_2m:kgm3	72	0.000774	float	
ceiling_height_agl:m	16489	0.177257	float	
clear_sky_energy_1h:J	78	0.000839	float	
clear_sky_rad:W	72	0.000774	float	
cloud_base_agl:m	6810	0.073208	float	
dew_point_2m:K	72	0.000774	float	
diffuse_rad:W	72	0.000774	float	
diffuse_rad_1h:J	78	0.000839	float	
direct_rad:W	72	0.000774	float	
direct_rad_1h:J	78	0.000839	float	
effective_cloud_cover:p	72	0.000774	float	
elevation:m	72	0.000774	float	
fresh_snow_1h:cm	78	0.000839	float	
fresh_snow_3h:cm	78	0.000839	float	
fresh_snow_6h:cm	78	0.000839	float	
hour			int	
is_day:idx	72	0.000774	float	
is_estimated			int	
is_in_shadow:idx	72	0.000774	float	
location			object	

msl_pressure:hPa	72	0.000774	float
precip_5min:mm	72	0.000774	float
precip_type_5min:idx	72	0.000774	float
pressure_100m:hPa	72	0.000774	float
pressure_50m:hPa	72	0.000774	float
rain_water:kgm2	72	0.000774	float
relative_humidity_1000hPa:p	72	0.000774	float
sample_weight			float
sfc_pressure:hPa	72	0.000774	float
snow_depth:cm	72	0.000774	float
snow_melt_10min:mm	72	0.000774	float
snow_water:kgm2	72	0.000774	float
sun_azimuth:d	72	0.000774	float
sun_elevation:d	72	0.000774	float
super_cooled_liquid_water:kgm2	72	0.000774	float
t_1000hPa:K	72	0.000774	float
total_cloud_cover:p	72	0.000774	float
visibility:m	72	0.000774	float
wind_speed_10m:ms	72	0.000774	float
wind_speed_u_10m:ms	72	0.000774	float
wind_speed_v_10m:ms	72	0.000774	float
y			float

	variable_type	special_types
absolute_humidity_2m:gm3	numeric	
air_density_2m:kgm3	numeric	
ceiling_height_agl:m	numeric	
clear_sky_energy_1h:J	numeric	
clear_sky_rad:W	numeric	
cloud_base_agl:m	numeric	
dew_point_2m:K	numeric	
diffuse_rad:W	numeric	
diffuse_rad_1h:J	numeric	
direct_rad:W	numeric	
direct_rad_1h:J	numeric	
effective_cloud_cover:p	numeric	
elevation:m	category	
fresh_snow_1h:cm	numeric	
fresh_snow_3h:cm	numeric	
fresh_snow_6h:cm	numeric	
hour	numeric	
is_day:idx	category	
is_estimated	category	
is_in_shadow:idx	category	
location	category	
msl_pressure:hPa	numeric	
precip_5min:mm	numeric	
precip_type_5min:idx	category	

pressure_100m:hPa	numeric
pressure_50m:hPa	numeric
rain_water:kgm2	numeric
relative_humidity_1000hPa:p	numeric
sample_weight	category
sfc_pressure:hPa	numeric
snow_depth:cm	numeric
snow_melt_10min:mm	numeric
snow_water:kgm2	numeric
sun_azimuth:d	numeric
sun_elevation:d	numeric
super_cooled_liquid_water:kgm2	numeric
t_1000hPa:K	numeric
total_cloud_cover:p	numeric
visibility:m	numeric
wind_speed_10m:ms	numeric
wind_speed_u_10m:ms	numeric
wind_speed_v_10m:ms	numeric
y	numeric

test_data dataset summary

	count	unique	top	freq	mean \
absolute_humidity_2m:gm3	5791	289			4.192639
air_density_2m:kgm3	5791	640			1.280018
ceiling_height_agl:m	4395	4247			3278.267059
clear_sky_energy_1h:J	5788	3059			469132.824948
clear_sky_rad:W	5791	2046			130.246477
cloud_base_agl:m	4934	4719			1733.271034
dew_point_2m:K	5791	948			270.733081
diffuse_rad:W	5791	2237			42.175259
diffuse_rad_1h:J	5788	3065			152461.828645
direct_rad:W	5791	1829			51.829421
direct_rad_1h:J	5788	2676			186526.762509
effective_cloud_cover:p	5791	2100			66.598541
elevation:m	5791	3			11.262131
fresh_snow_1h:cm	5788	23			0.032308
fresh_snow_3h:cm	5788	42			0.100259
fresh_snow_6h:cm	5788	60			0.204492
hour	5791	24			11.499396
is_day:idx	5791	5			0.488387
is_estimated	5791	1			1.0
is_in_shadow:idx	5791	5			0.555085
location	5791	3	A	2161	
msl_pressure:hPa	5791	2040			1012.678587
precip_5min:mm	5791	63			0.003687
precip_type_5min:idx	5791	12			0.086039
pressure_100m:hPa	5791	2124			998.781639
pressure_50m:hPa	5791	2134			1005.02648

rain_water:kgm2	5791	7	0.000984
relative_humidity_1000hPa:p	5791	2051	70.810205
sample_weight	5791	1	2.0
sfc_pressure:hPa	5791	2148	1011.29959
snow_depth:cm	5791	78	0.131661
snow_melt_10min:mm	5791	38	0.000695
snow_water:kgm2	5791	68	0.078393
sun_azimuth:d	5791	5681	179.475343
sun_elevation:d	5791	5093	-0.927197
super_cooled_liquid_water:kgm2	5791	31	0.035175
t_1000hPa:K	5791	825	275.185991
total_cloud_cover:p	5791	1838	71.785616
visibility:m	5791	5784	29884.461577
wind_speed_10m:ms	5791	424	3.227599
wind_speed_u_10m:ms	5791	672	0.668019
wind_speed_v_10m:ms	5791	483	0.538344
y	5791	2304	272.991992

	std	min	25% \
absolute_humidity_2m:gm3	1.300644	1.1	3.35
air_density_2m:kgm3	0.024372	1.219	1.26375
ceiling_height_agl:m	2590.751931	27.925	1149.0625
clear_sky_energy_1h:J	689638.596662	0.0	0.0
clear_sky_rad:W	191.578221	0.0	0.0
cloud_base_agl:m	1987.046511	27.5	525.4375
dew_point_2m:K	4.634046	255.05	268.33749
diffuse_rad:W	59.158733	0.0	0.0
diffuse_rad_1h:J	211011.771342	0.0	0.0
direct_rad:W	110.450287	0.0	0.0
direct_rad_1h:J	393513.65175	0.0	0.0
effective_cloud_cover:p	37.583548	0.0	33.6375
elevation:m	7.8114	6.0	6.0
fresh_snow_1h:cm	0.170919	0.0	0.0
fresh_snow_3h:cm	0.425766	0.0	0.0
fresh_snow_6h:cm	0.738932	0.0	0.0
hour	6.920293	0.0	6.0
is_day:idx	0.486436	0.0	0.0
is_estimated	0.0	1.0	1.0
is_in_shadow:idx	0.483636	0.0	0.0
location			
msl_pressure:hPa	13.953847	975.3	1003.875
precip_5min:mm	0.017701	0.0	0.0
precip_type_5min:idx	0.393918	0.0	0.0
pressure_100m:hPa	13.825369	962.4	989.9
pressure_50m:hPa	13.873049	968.45	996.087475
rain_water:kgm2	0.009596	0.0	0.0
relative_humidity_1000hPa:p	14.940249	21.325	60.75
sample_weight	0.0	2.0	2.0

sfc_pressure:hPa	13.921629	974.55	1002.25
snow_depth:cm	0.635847	0.0	0.0
snow_melt_10min:mm	0.007333	0.0	0.0
snow_water:kgm2	0.189057	0.0	0.0
sun_azimuth:d	96.891969	14.913	94.264625
sun_elevation:d	20.775858	-44.28175	-17.109625
super_cooled_liquid_water:kgm2	0.084895	0.0	0.0
t_1000hPa:K	3.823552	261.975	272.8
total_cloud_cover:p	37.578218	0.0	41.8
visibility:m	14669.627165	1215.4	18727.05
wind_speed_10m:ms	1.869023	0.05	1.725
wind_speed_u_10m:ms	3.12501	-7.15	-1.75
wind_speed_v_10m:ms	1.838513	-5.3	-0.8
y	770.841016	-0.0	0.0

	50%	75%	max	dtypes \
absolute_humidity_2m:gm3	4.3	5.05	7.7	float64
air_density_2m:kgm3	1.279	1.29375	1.37175	float64
ceiling_height_agl:m	2618.95	4661.025	12294.901	float64
clear_sky_energy_1h:J	11008.5	791394.0	2554290.5	float64
clear_sky_rad:W	2.675	221.925	710.5	float64
cloud_base_agl:m	904.825	2014.962525	10674.3	float64
dew_point_2m:K	271.6	273.9	280.4	float64
diffuse_rad:W	1.775	78.4875	311.95	float64
diffuse_rad_1h:J	18860.9	279202.425	1071799.5	float64
direct_rad:W	0.0	34.0875	530.15	float64
direct_rad_1h:J	0.0	129529.5	1895533.0	float64
effective_cloud_cover:p	85.375	99.975	100.0	float64
elevation:m	7.0	24.0	24.0	float64
fresh_snow_1h:cm	0.0	0.0	2.6	float64
fresh_snow_3h:cm	0.0	0.0	5.2	float64
fresh_snow_6h:cm	0.0	0.0	7.5	float64
hour	11.0	17.0	23.0	int64
is_day:idx	0.25	1.0	1.0	float64
is_estimated	1.0	1.0	1.0	int64
is_in_shadow:idx	1.0	1.0	1.0	float64
location				object
msl_pressure:hPa	1011.625	1023.8125	1041.3501	float64
precip_5min:mm	0.0	0.0	0.2475	float64
precip_type_5min:idx	0.0	0.0	3.0	float64
pressure_100m:hPa	997.9	1009.875	1028.05	float64
pressure_50m:hPa	1004.1	1016.1625	1034.45	float64
rain_water:kgm2	0.0	0.0	0.175	float64
relative_humidity_1000hPa:p	73.1	82.075	98.0	float64
sample_weight	2.0	2.0	2.0	int64
sfc_pressure:hPa	1010.35	1022.5125	1040.8501	float64
snow_depth:cm	0.0	0.0	4.9	float64
snow_melt_10min:mm	0.0	0.0	0.14	float64

snow_water:kgm2	0.0	0.1	2.15	float64
sun_azimuth:d	179.52899	263.49875	347.37848	float64
sun_elevation:d	-0.79825	15.30325	41.13025	float64
super_cooled_liquid_water:kgm2	0.0	0.0	0.75	float64
t_1000hPa:K	275.175	277.525	285.1	float64
total_cloud_cover:p	96.65	100.0	100.0	float64
visibility:m	31311.025	40438.6635	66178.45	float64
wind_speed_10m:ms	2.9	4.45	10.2	float64
wind_speed_u_10m:ms	0.3	2.9	9.95	float64
wind_speed_v_10m:ms	0.625	1.825	7.15	float64
y	0.0	142.906699	5172.64	float64

	missing_count	missing_ratio	raw_type	\
absolute_humidity_2m:gm3			float	
air_density_2m:kgm3			float	
ceiling_height_agl:m	1396	0.241064	float	
clear_sky_energy_1h:J	3	0.000518	float	
clear_sky_rad:W			float	
cloud_base_agl:m	857	0.147988	float	
dew_point_2m:K			float	
diffuse_rad:W			float	
diffuse_rad_1h:J	3	0.000518	float	
direct_rad:W			float	
direct_rad_1h:J	3	0.000518	float	
effective_cloud_cover:p			float	
elevation:m			float	
fresh_snow_1h:cm	3	0.000518	float	
fresh_snow_3h:cm	3	0.000518	float	
fresh_snow_6h:cm	3	0.000518	float	
hour			int	
is_day:idx			float	
is_estimated			int	
is_in_shadow:idx			float	
location			object	
msl_pressure:hPa			float	
precip_5min:mm			float	
precip_type_5min:idx			float	
pressure_100m:hPa			float	
pressure_50m:hPa			float	
rain_water:kgm2			float	
relative_humidity_1000hPa:p			float	
sample_weight			int	
sfc_pressure:hPa			float	
snow_depth:cm			float	
snow_melt_10min:mm			float	
snow_water:kgm2			float	
sun_azimuth:d			float	
sun_elevation:d			float	

super_cooled_liquid_water:kgm2	float
t_1000hPa:K	float
total_cloud_cover:p	float
visibility:m	float
wind_speed_10m:ms	float
wind_speed_u_10m:ms	float
wind_speed_v_10m:ms	float
y	float

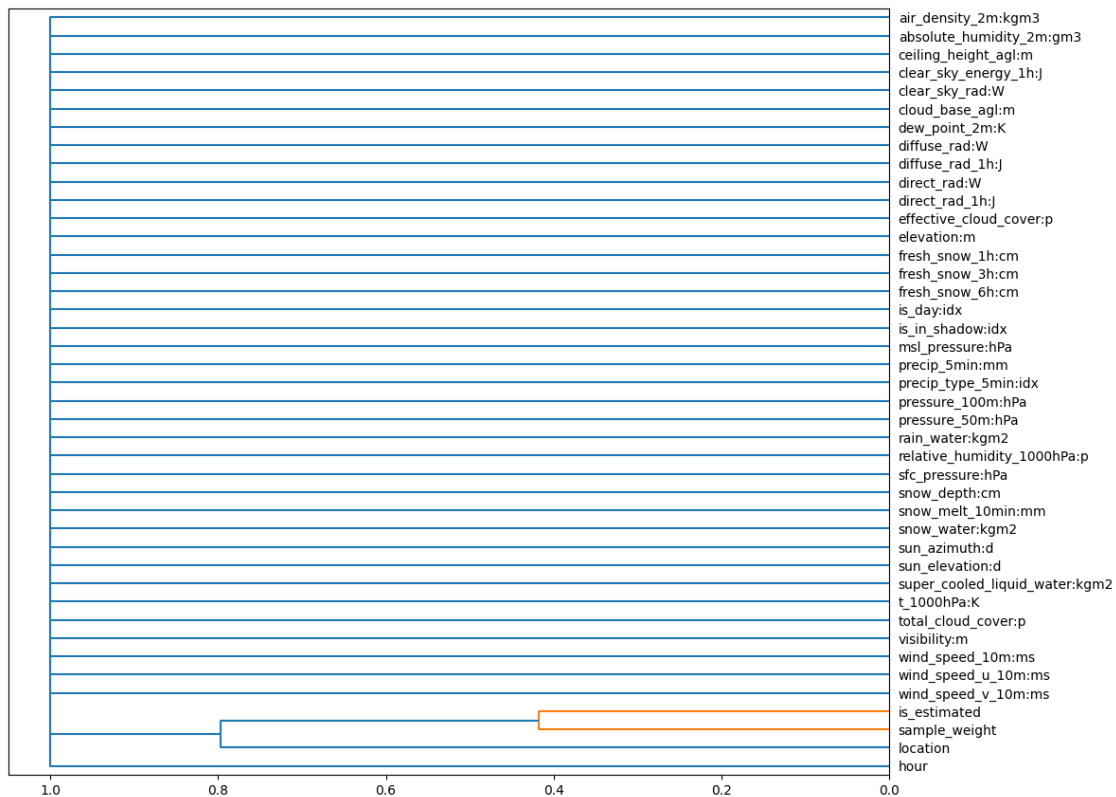
	variable_type	special_types
absolute_humidity_2m:gm3	numeric	
air_density_2m:kgm3	numeric	
ceiling_height_agl:m	numeric	
clear_sky_energy_1h:J	numeric	
clear_sky_rad:W	numeric	
cloud_base_agl:m	numeric	
dew_point_2m:K	numeric	
diffuse_rad:W	numeric	
diffuse_rad_1h:J	numeric	
direct_rad:W	numeric	
direct_rad_1h:J	numeric	
effective_cloud_cover:p	numeric	
elevation:m	category	
fresh_snow_1h:cm	numeric	
fresh_snow_3h:cm	numeric	
fresh_snow_6h:cm	numeric	
hour	numeric	
is_day:idx	category	
is_estimated	category	
is_in_shadow:idx	category	
location	category	
msl_pressure:hPa	numeric	
precip_5min:mm	numeric	
precip_type_5min:idx	category	
pressure_100m:hPa	numeric	
pressure_50m:hPa	numeric	
rain_water:kgm2	category	
relative_humidity_1000hPa:p	numeric	
sample_weight	category	
sfc_pressure:hPa	numeric	
snow_depth:cm	numeric	
snow_melt_10min:mm	numeric	
snow_water:kgm2	numeric	
sun_azimuth:d	numeric	
sun_elevation:d	numeric	
super_cooled_liquid_water:kgm2	numeric	
t_1000hPa:K	numeric	
total_cloud_cover:p	numeric	

visibility:m	numeric
wind_speed_10m:ms	numeric
wind_speed_u_10m:ms	numeric
wind_speed_v_10m:ms	numeric
y	numeric

Types warnings summary

	train_data	test_data	warnings
sample_weight	float	int	warning

1.0.1 Feature Distance

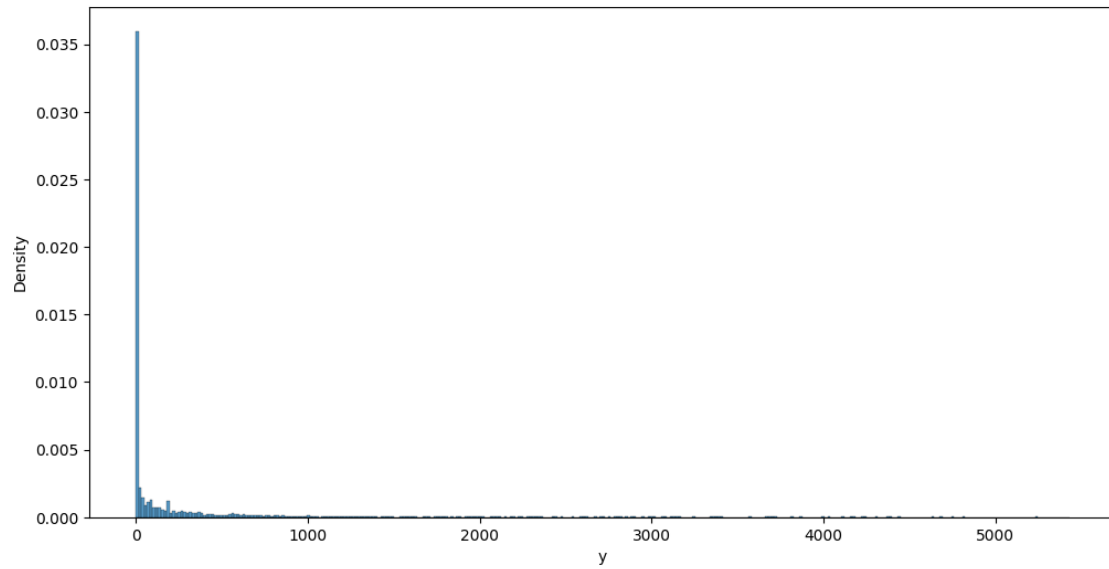


```
[11]: if run_analysis:
      auto.target_analysis(train_data=train_data, label="y")
```

1.1 Target variable analysis

	count	mean	std	min	25%	50%	75%	max	dtypes \
y	10000	283.88153	748.627904	0.0	0.0	0.0	176.4	5428.72	float64

	unique	missing_count	missing_ratio	raw_type	special_types
y	2537			float	

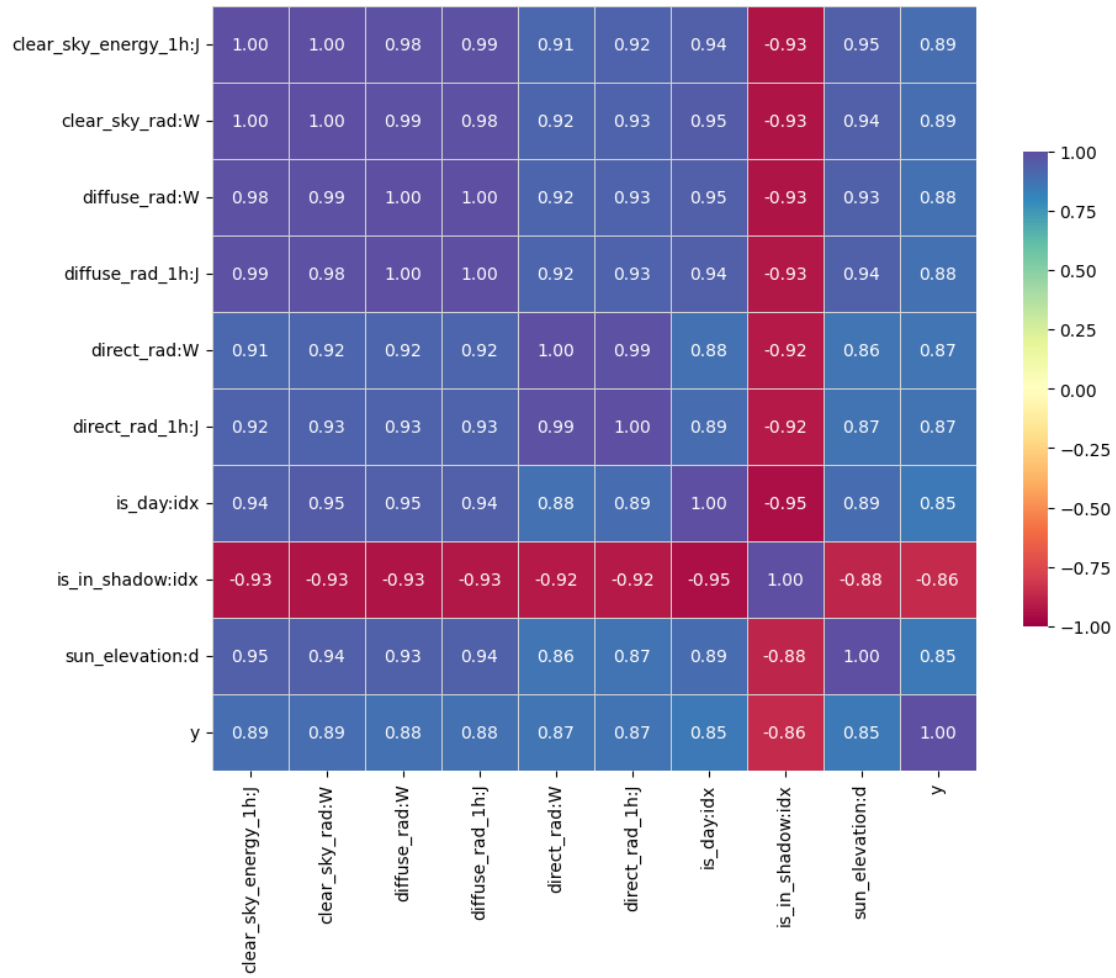


1.1.1 Distribution fits for target variable

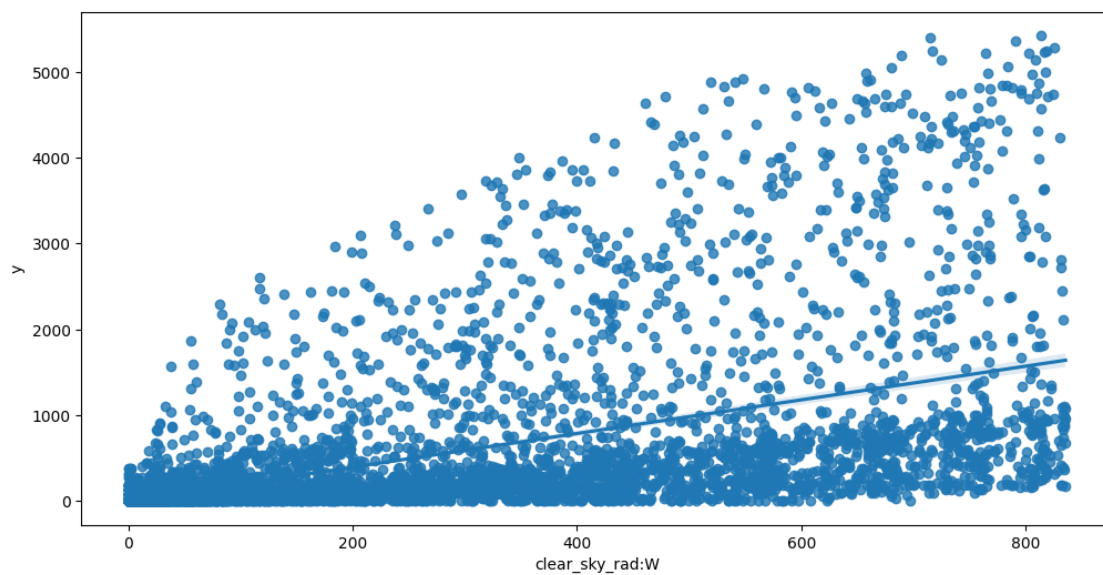
- none of the [attempted](#) distribution fits satisfy specified minimum p-value threshold: 0.01

1.1.2 Target variable correlations

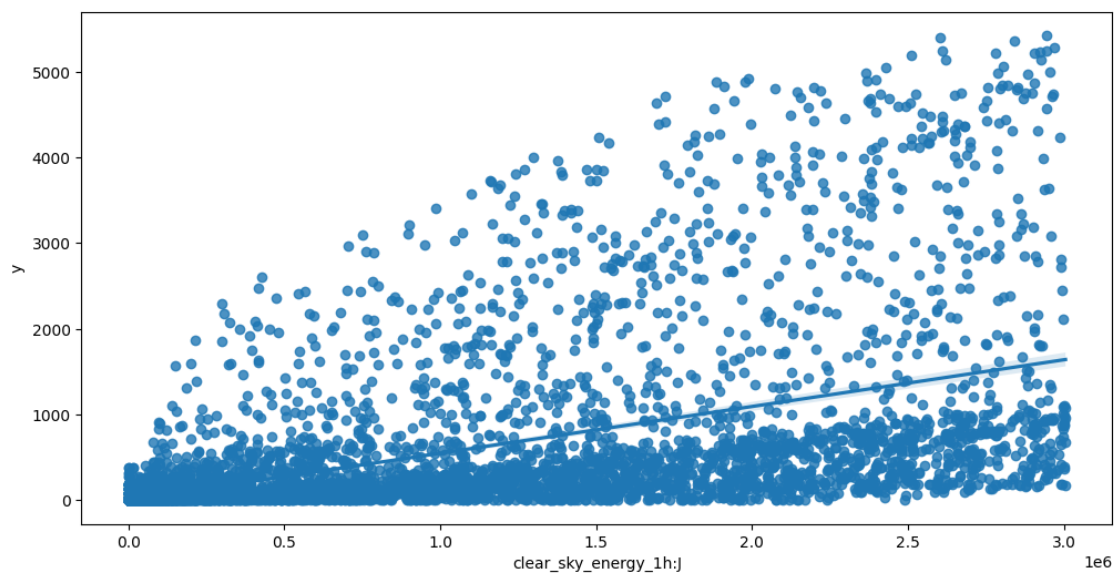
`train_data` - spearman correlation matrix; focus: absolute correlation for $y \geq 0.5$
(sample size: 10000)



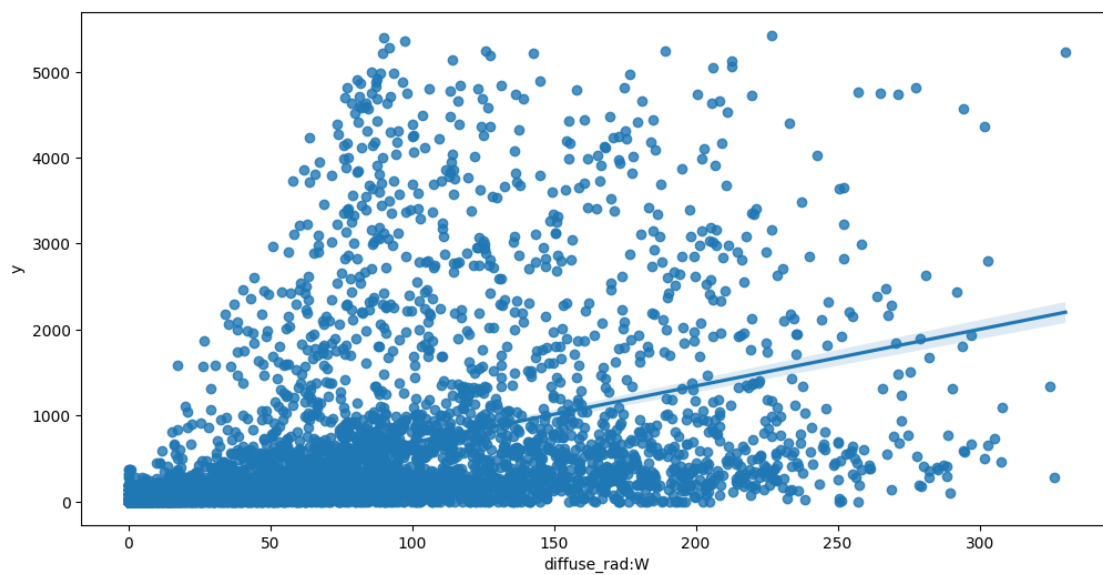
Feature interaction between `clear_sky_rad:W/y` in `train_data` (sample size: 10000)



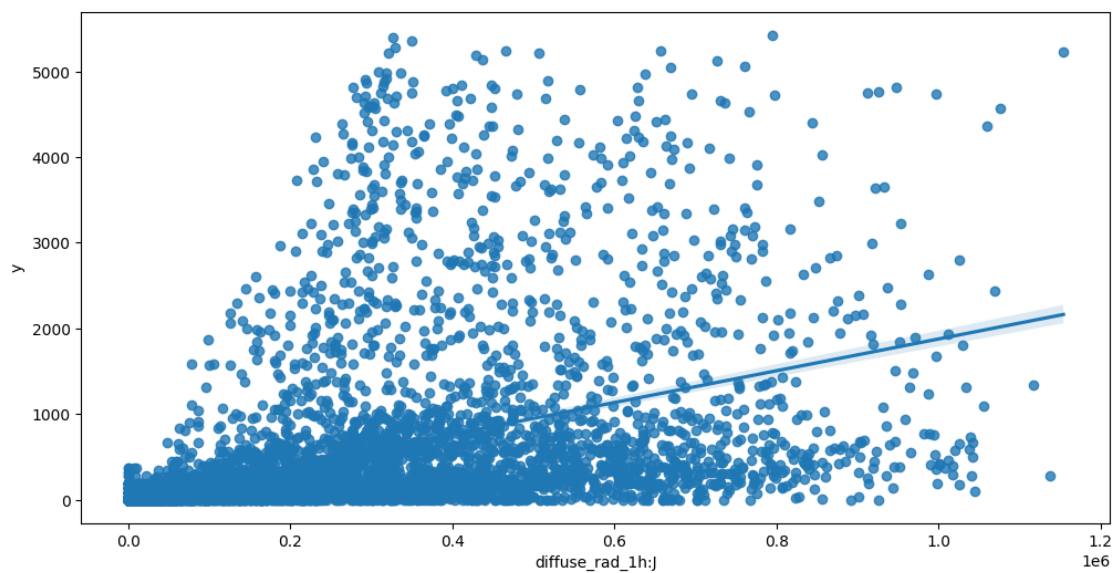
Feature interaction between clear_sky_energy_1h:J/y in train_data (sample size: 10000)



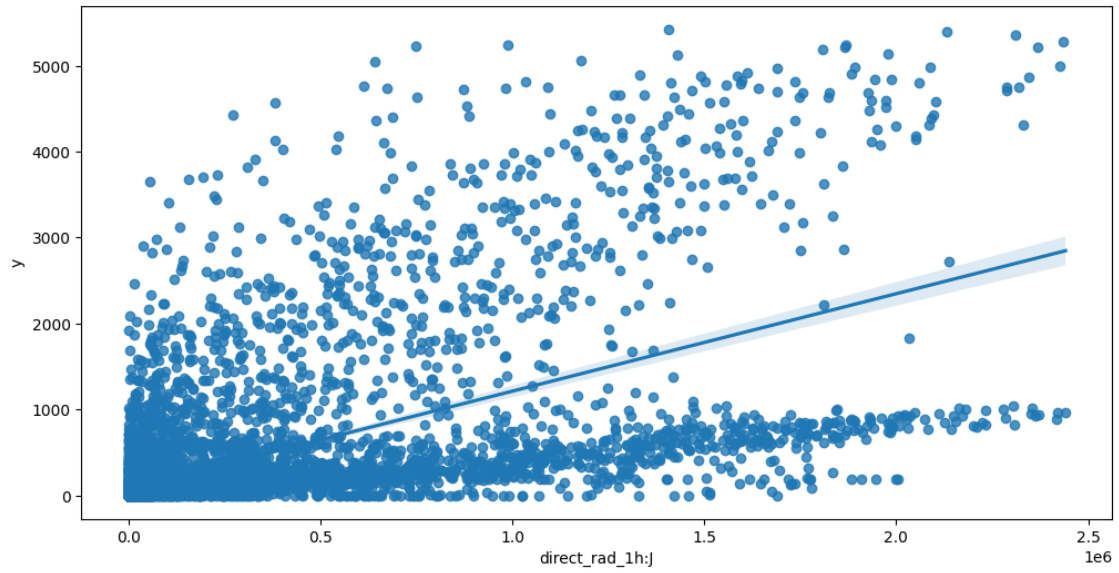
Feature interaction between diffuse_rad:W/y in train_data (sample size: 10000)



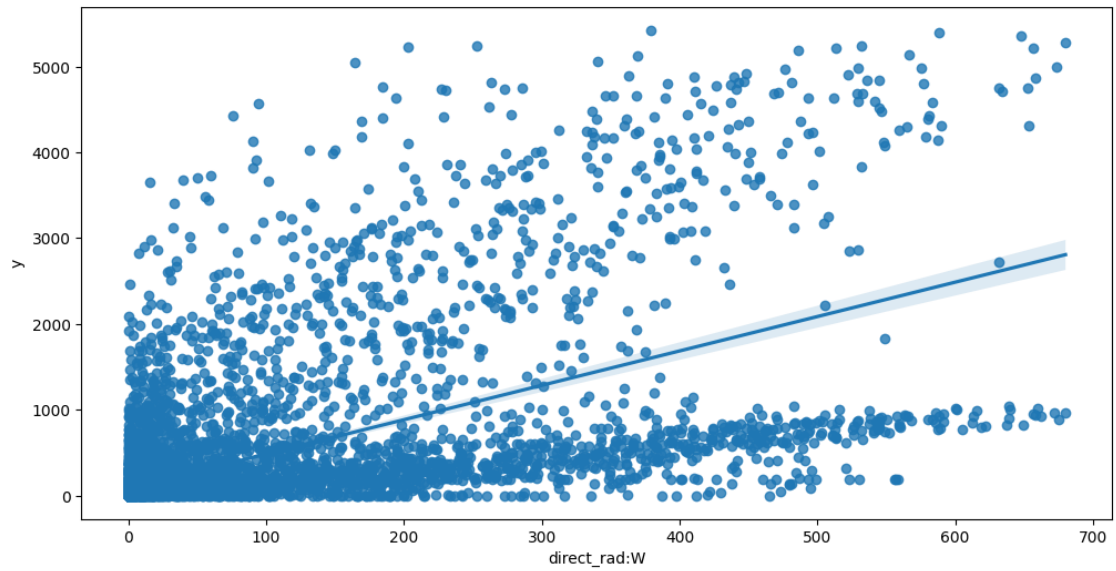
Feature interaction between diffuse_rad_1h:J/y in train_data (sample size: 10000)



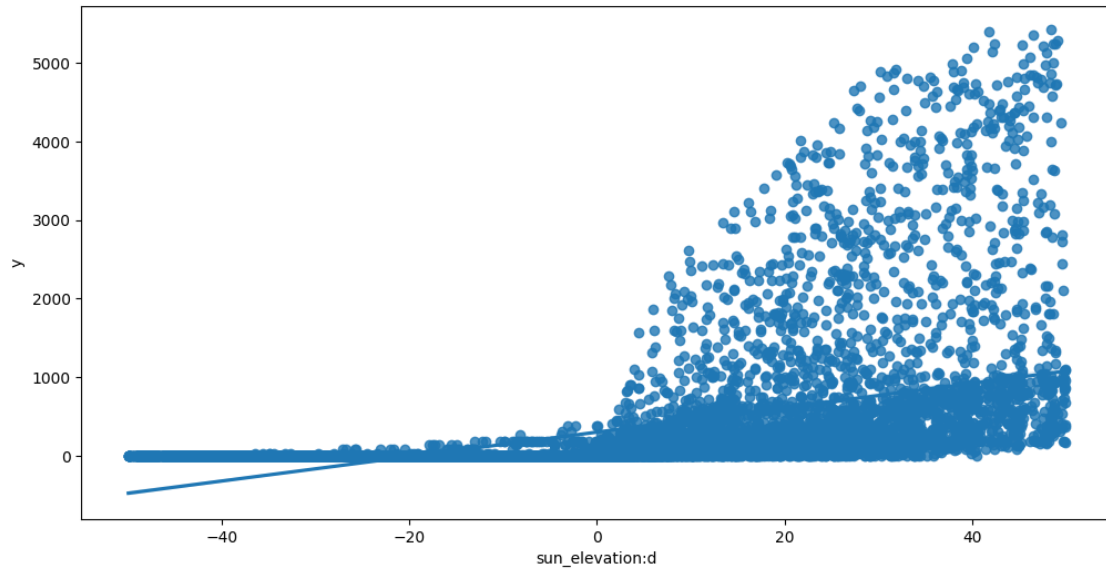
Feature interaction between direct_rad_1h:J/y in train_data (sample size: 10000)



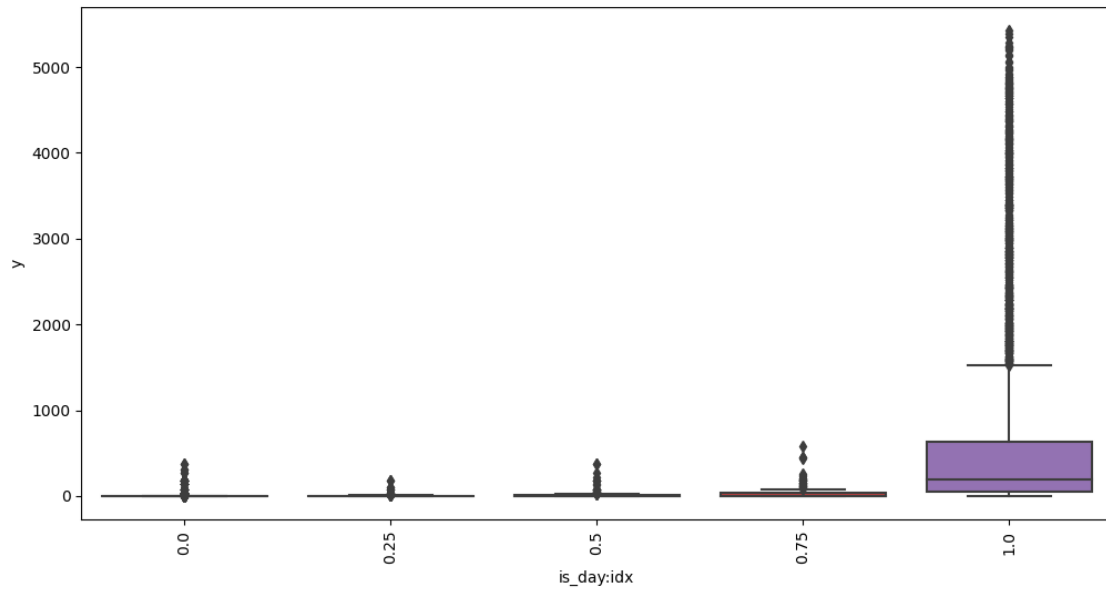
Feature interaction between direct_rad:W/y in train_data (sample size: 10000)



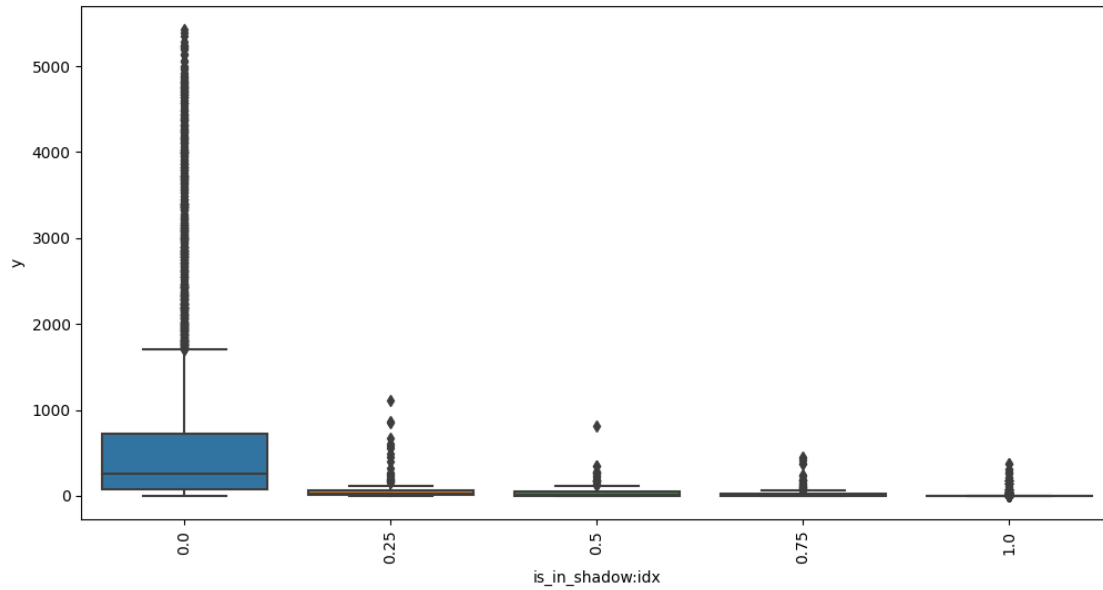
Feature interaction between sun_elevation:d/y in train_data (sample size: 10000)



Feature interaction between `is_day:idx/y` in `train_data` (sample size: 10000)



Feature interaction between `is_in_shadow:idx/y` in `train_data` (sample size: 10000)



2 Starting

```
[12]: import os

# Get the last submission number
last_submission_number = int(max([int(filename.split('_')[1].split('.')[0]) for
    ↪ filename in os.listdir('submissions') if "submission" in filename]))
print("Last submission number:", last_submission_number)
print("Now creating submission number:", last_submission_number + 1)

# Create the new filename
new_filename = f'submission_{last_submission_number + 1}'

hello = os.environ.get('HELLO')
if hello is not None:
    new_filename += f'_{hello}'

print("New filename:", new_filename)
```

```
Last submission number: 90
Now creating submission number: 91
New filename: submission_91
```

```
[13]: predictors = [None, None, None]
```

```

[ ]: def fit_predictor_for_location(loc):
    print(f"Training model for location {loc}...")
    # sum of sample weights for this location, and number of rows, for both
    ↪train and tune data and test data
    print("Train data sample weight sum:", train_data[train_data["location"] ==
    ↪loc]["sample_weight"].sum())
    print("Train data number of rows:", train_data[train_data["location"] ==
    ↪loc].shape[0])
    if use_tune_data:
        print("Tune data sample weight sum:",
    ↪tuning_data[tuning_data["location"] == loc]["sample_weight"].sum())
        print("Tune data number of rows:", tuning_data[tuning_data["location"]
    ↪== loc].shape[0])
    if use_test_data:
        print("Test data sample weight sum:", test_data[test_data["location"]
    ↪== loc]["sample_weight"].sum())
        print("Test data number of rows:", test_data[test_data["location"] ==
    ↪loc].shape[0])
    predictor = TabularPredictor(
        label=label,
        eval_metric=metric,
        path=f"AutogluonModels/{new_filename}_{loc}",
        sample_weight=sample_weight,
        weight_evaluation=weight_evaluation,
        groups="group" if use_groups else None,
    ).fit(
        train_data=train_data[train_data["location"] == loc],
        time_limit=time_limit,
        presets=presets,
        num_stack_levels=num_stack_levels,
        num_bag_folds=num_bag_folds if not use_groups else 2, # just put
    ↪somethin, will be overwritten anyways
        tuning_data=tuning_data[tuning_data["location"] == loc] if
    ↪use_tune_data else None,
        use_bag_holdout=use_bag_holdout,
        holdout_frac=holdout_frac,
    )

    # evaluate on test data
    if use_test_data:
        # drop sample_weight column
        t = test_data[test_data["location"] == loc]#.
    ↪drop(columns=["sample_weight"])
        perf = predictor.evaluate(t)
        print("Evaluation on test data:")
        print(perf[predictor.eval_metric.name])

```

```

    return predictor

loc = "A"
predictors[0] = fit_predictor_for_location(loc)

```

Presets specified: ['best_quality']

Stack configuration (auto_stack=True): num_stack_levels=2, num_bag_folds=8, num_bag_sets=20

Values in column 'sample_weight' used as sample weights instead of predictive features. Evaluation will report weighted metrics, so ensure same column exists in test data.

Beginning AutoGluon training ... Time limit = 7200s

AutoGluon will save models to "AutogluonModels/submission_91_A/"

AutoGluon Version: 0.8.2

Python Version: 3.10.12

Operating System: Linux

Platform Machine: x86_64

Platform Version: #1 SMP Debian 5.10.197-1 (2023-09-29)

Disk Space Avail: 292.91 GB / 315.93 GB (92.7%)

Train Data Rows: 34085

Train Data Columns: 42

Label Column: y

Preprocessing data ...

AutoGluon infers your prediction problem is: 'regression' (because dtype of label-column == float and many unique label-values observed).

Label info (max, min, mean, stddev): (5733.42, 0.0, 630.59471, 1165.90242)

If 'regression' is not the correct problem_type, please manually specify the problem_type parameter during predictor init (You may specify problem_type as one of: ['binary', 'multiclass', 'regression'])

Using Feature Generators to preprocess the data ...

Fitting AutoMLPipelineFeatureGenerator...

Available Memory: 132175.85 MB

Train Data (Original) Memory Usage: 12.88 MB (0.0% of available memory)

Inferring data type of each feature based on column values. Set feature_metadata_in to manually specify special dtypes of the features.

Stage 1 Generators:

Fitting AsTypeFeatureGenerator...

Note: Converting 2 features to boolean dtype as they only contain 2 unique values.

Stage 2 Generators:

Fitting FillNaFeatureGenerator...

Stage 3 Generators:

Fitting IdentityFeatureGenerator...

Stage 4 Generators:

Fitting DropUniqueFeatureGenerator...

Training model for location A...
Train data sample weight sum: 34084.999999999985
Train data number of rows: 34085

Stage 5 Generators:

Fitting DropDuplicatesFeatureGenerator...

Useless Original Features (Count: 1): ['location']

These features carry no predictive signal and should be manually investigated.

This is typically a feature which has the same value for all rows.

These features do not need to be present at inference time.

Types of features in original data (raw dtype, special dtypes):

('float', []) : 38 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', ...]

('int', []) : 2 | ['is_estimated', 'hour']

Types of features in processed data (raw dtype, special dtypes):

('float', []) : 37 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', ...]

('int', []) : 1 | ['hour']

('int', ['bool']) : 2 | ['elevation:m', 'is_estimated']

0.1s = Fit runtime

40 features in original data used to generate 40 features in processed data.

Train Data (Processed) Memory Usage: 10.43 MB (0.0% of available memory)

Data preprocessing and feature engineering runtime = 0.18s ...

AutoGluon will gauge predictive performance using evaluation metric:

'mean_absolute_error'

This metric's sign has been flipped to adhere to being higher_is_better. The metric score can be multiplied by -1 to get the metric value.

To change this, specify the eval_metric parameter of Predictor()

User-specified model hyperparameters to be fit:

```
{
    'NN_TORCH': {},
    'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {}],
    'GBMLarge': {},
    'CAT': {},
    'XGB': {},
    'FASTAI': {},
    'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
    'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
```

```

{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}},
    'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
AutoGluon will fit 3 stack levels (L1 to L3) ...
Fitting 11 L1 models ...
Fitting model: KNeighborsUnif_BAG_L1 ... Training model for up to 3199.12s of
the 7199.82s of remaining time.
    -216.24 = Validation score    (-mean_absolute_error)
    0.04s   = Training    runtime
    0.4s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 ... Training model for up to 3198.57s of
the 7199.27s of remaining time.
    -217.2002 = Validation score    (-mean_absolute_error)
    0.04s     = Training    runtime
    0.41s     = Validation runtime
Fitting model: LightGBMXT_BAG_L1 ... Training model for up to 3198.07s of the
7198.77s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -144.3217 = Validation score    (-mean_absolute_error)
    39.27s    = Training    runtime
    19.74s    = Validation runtime
Fitting model: LightGBM_BAG_L1 ... Training model for up to 3149.64s of the
7150.33s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -154.9519 = Validation score    (-mean_absolute_error)
    45.99s    = Training    runtime
    19.51s    = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 ... Training model for up to 3100.25s of
the 7100.95s of remaining time.
    -168.8812 = Validation score    (-mean_absolute_error)
    10.78s    = Training    runtime
    1.43s     = Validation runtime
Fitting model: CatBoost_BAG_L1 ... Training model for up to 3087.38s of the
7088.08s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -161.7348 = Validation score    (-mean_absolute_error)
    210.57s   = Training    runtime
    0.13s     = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 ... Training model for up to 2875.79s of the
6876.48s of remaining time.
    -169.8924 = Validation score    (-mean_absolute_error)
    2.16s     = Training    runtime

```

```

1.43s      = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 ... Training model for up to 2871.53s of
the 6872.23s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -173.9881      = Validation score    (-mean_absolute_error)
    41.79s      = Training    runtime
    0.65s      = Validation runtime
Fitting model: XGBoost_BAG_L1 ... Training model for up to 2828.62s of the
6829.32s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -164.8785      = Validation score    (-mean_absolute_error)
    64.66s      = Training    runtime
    4.85s      = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 ... Training model for up to 2760.45s of
the 6761.14s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -160.3243      = Validation score    (-mean_absolute_error)
    126.45s     = Training    runtime
    0.37s      = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 ... Training model for up to 2632.7s of the
6633.4s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -151.5461      = Validation score    (-mean_absolute_error)
    143.83s     = Training    runtime
    23.03s     = Validation runtime
Repeating k-fold bagging: 2/20
Fitting model: LightGBMXT_BAG_L1 ... Training model for up to 2483.75s of the
6484.45s of remaining time.
    Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -141.0915      = Validation score    (-mean_absolute_error)
    78.03s      = Training    runtime
    40.83s      = Validation runtime
Fitting model: LightGBM_BAG_L1 ... Training model for up to 2440.34s of the
6441.03s of remaining time.
    Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -150.7559      = Validation score    (-mean_absolute_error)
    92.34s      = Training    runtime
    36.82s      = Validation runtime
Fitting model: CatBoost_BAG_L1 ... Training model for up to 2390.73s of the
6391.43s of remaining time.
    Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy

```



```

-157.998          = Validation score    (-mean_absolute_error)
420.05s = Training    runtime
0.26s   = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 ... Training model for up to 2180.02s of
the 6180.72s of remaining time.
    Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
-170.7462          = Validation score    (-mean_absolute_error)
84.37s  = Training    runtime
1.33s   = Validation runtime
Fitting model: XGBoost_BAG_L1 ... Training model for up to 2136.17s of the
6136.87s of remaining time.
    Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
-160.2333          = Validation score    (-mean_absolute_error)
135.2s  = Training    runtime
11.0s   = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 ... Training model for up to 2062.87s of
the 6063.56s of remaining time.
    Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
-156.8688          = Validation score    (-mean_absolute_error)
239.65s = Training    runtime
0.76s   = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 ... Training model for up to 1948.46s of the
5949.15s of remaining time.
    Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
-149.0872          = Validation score    (-mean_absolute_error)
287.51s = Training    runtime
45.17s  = Validation runtime
Repeating k-fold bagging: 3/20
Fitting model: LightGBMXT_BAG_L1 ... Training model for up to 1799.11s of the
5799.8s of remaining time.
    Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy
-140.2858          = Validation score    (-mean_absolute_error)
117.35s = Training    runtime
59.43s  = Validation runtime
Fitting model: LightGBM_BAG_L1 ... Training model for up to 1755.46s of the
5756.15s of remaining time.
    Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy
-149.7701          = Validation score    (-mean_absolute_error)
139.18s = Training    runtime
50.17s  = Validation runtime
Fitting model: CatBoost_BAG_L1 ... Training model for up to 1705.76s of the
5706.46s of remaining time.

```

```

    Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -156.941      = Validation score    (-mean_absolute_error)
    630.37s      = Training    runtime
    0.4s         = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 ... Training model for up to 1494.27s of
the 5494.96s of remaining time.
    Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -169.6431     = Validation score    (-mean_absolute_error)
    126.47s      = Training    runtime
    1.96s        = Validation runtime
Fitting model: XGBoost_BAG_L1 ... Training model for up to 1450.8s of the
5451.5s of remaining time.
    Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -158.8005     = Validation score    (-mean_absolute_error)
    198.17s      = Training    runtime
    12.93s       = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 ... Training model for up to 1385.03s of
the 5385.73s of remaining time.
    Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy

```

```
[ ]: loc = "B"
predictors[1] = fit_predictor_for_location(loc)
```

```
[ ]: loc = "C"
predictors[2] = fit_predictor_for_location(loc)
```

3 Submit

```
[ ]: import pandas as pd
import matplotlib.pyplot as plt

train_data_with_dates = TabularDataset('X_train_raw.csv')
train_data_with_dates["ds"] = pd.to_datetime(train_data_with_dates["ds"])

test_data = TabularDataset('X_test_raw.csv')
test_data["ds"] = pd.to_datetime(test_data["ds"])
#test_data

```

```
[ ]: test_ids = TabularDataset('test.csv')
test_ids["time"] = pd.to_datetime(test_ids["time"])
# merge test_data with test_ids
test_data_merged = pd.merge(test_data, test_ids, how="inner", right_on=["time",
↵ "location"], left_on=["ds", "location"])
```

```
#test_data_merged
```

```
[ ]: # predict, grouped by location
predictions = []
location_map = {
    "A": 0,
    "B": 1,
    "C": 2
}
for loc, group in test_data.groupby('location'):
    i = location_map[loc]
    subset = test_data_merged[test_data_merged["location"] == loc].
    ↪reset_index(drop=True)
    #print(subset)
    pred = predictors[i].predict(subset)
    subset["prediction"] = pred
    predictions.append(subset)

    # get past predictions
    past_pred = predictors[i].
    ↪predict(train_data_with_dates[train_data_with_dates["location"] == loc])
    train_data_with_dates.loc[train_data_with_dates["location"] == loc,
    ↪"prediction"] = past_pred
```

```
[ ]: # plot predictions for location A, in addition to train data for A
for loc, idx in location_map.items():
    fig, ax = plt.subplots(figsize=(20, 10))
    # plot train data
    train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',
    ↪y='y', ax=ax, label="train data")

    # plot predictions
    predictions[idx].plot(x='ds', y='prediction', ax=ax, label="predictions")

    # plot past predictions
    train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',
    ↪y='prediction', ax=ax, label="past predictions")

    # title
    ax.set_title(f"Predictions for location {loc}")
```

```
[ ]: # concatenate predictions
submissions_df = pd.concat(predictions)
submissions_df = submissions_df[["id", "prediction"]]
submissions_df
```

```
[ ]: # Save the submission DataFrame to submissions folder, create new name based on
      ↪ last submission, format is submission_<last_submission_number + 1>.csv

      # Save the submission
      print(f"Saving submission to submissions/{new_filename}.csv")
      submissions_df.to_csv(os.path.join('submissions', f"{new_filename}.csv"),
        ↪ index=False)
      print("jall1a")
```

```
[ ]: # save this running notebook
      from IPython.display import display, Javascript
      import time

      # hei123

      display(Javascript("IPython.notebook.save_checkpoint();"))

      time.sleep(3)
```

```
[ ]: # save this notebook to submissions folder
      import subprocess
      import os
      subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
        ↪ join('notebook_pdfs', f"{new_filename}.pdf"), "autogluon_each_location.
        ↪ ipynb"])
```

```
[ ]: # feature importance
      location="A"
      split_time = pd.Timestamp("2022-10-28 22:00:00")
      estimated = train_data_with_dates[train_data_with_dates["ds"] >= split_time]
      estimated = estimated[estimated["location"] == location]
      predictors[0].feature_importance(feature_stage="original", data=estimated,
        ↪ time_limit=60*10)
```

```
[ ]: # feature importance
      observed = train_data_with_dates[train_data_with_dates["ds"] < split_time]
      observed = observed[observed["location"] == location]
      predictors[0].feature_importance(feature_stage="original", data=observed,
        ↪ time_limit=60*10)
```

```
[ ]: display(Javascript("IPython.notebook.save_checkpoint();"))
      time.sleep(3)

      subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
        ↪ join('notebook_pdfs', f"{new_filename}_with_feature_importance.pdf"),
        ↪ "autogluon_each_location.ipynb"])
```

```
[ ]: # import subprocess

# def execute_git_command(directory, command):
#     """Execute a Git command in the specified directory."""
#     try:
#         result = subprocess.check_output(['git', '-C', directory] + command,
#     ↪ stderr=subprocess.STDOUT)
#         return result.decode('utf-8').strip(), True
#     except subprocess.CalledProcessError as e:
#         print(f"Git command failed with message: {e.output.decode('utf-8').
#     ↪ strip()}")
#         return e.output.decode('utf-8').strip(), False

# git_repo_path = "."

# execute_git_command(git_repo_path, ['config', 'user.email',
#     ↪ 'henrikskog01@gmail.com'])
# execute_git_command(git_repo_path, ['config', 'user.name', 'hello if hello is
#     ↪ not None else 'Henrik eller Jørgen'])

# branch_name = new_filename

# # add datetime to branch name
# branch_name += f"_{pd.Timestamp.now().strftime('%Y-%m-%d_%H-%M-%S')}"

# commit_msg = "run result"

# execute_git_command(git_repo_path, ['checkout', '-b', branch_name])

# # Navigate to your repo and commit changes
# execute_git_command(git_repo_path, ['add', '.'])
# execute_git_command(git_repo_path, ['commit', '-m', commit_msg])

# # Push to remote
# output, success = execute_git_command(git_repo_path, ['push',
#     ↪ 'origin', branch_name])

# # If the push fails, try setting an upstream branch and push again
# if not success and 'upstream' in output:
#     print("Attempting to set upstream and push again...")
#     execute_git_command(git_repo_path, ['push', '--set-upstream',
#     ↪ 'origin', branch_name])
#     execute_git_command(git_repo_path, ['push', 'origin', 'henrik_branch'])

# execute_git_command(git_repo_path, ['checkout', 'main'])
```